AFRL-RI-RS-TR-2015-205

HIGH SPEED OBLIVIOUS RANDOM ACCESS MEMORY (HS-ORAM)

PRIVATE MACHINES, INC.

SEPTEMBER 2015

FINAL TECHNICAL REPORT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

STINFO COPY

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE

■ **AIR FORCE MATERIEL COMMAND** ■ **UNITED STATES AIR FORCE** ■ **ROME, NY 13441**

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report was cleared for public release by the Defense Advanced Research Projects Agency (DARPA) Public Release Center and is available to the general public, including foreign nationals. Copies may be obtained from the Defense Technical Information Center (DTIC) (http://www.dtic.mil).

AFRL-RI-RS-TR-2015-205 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE DIRECTOR:

/ S /
JONATHAN HEINER
Work Unit Manager

/ S /
MARK LINDERMAN
Technical Advisor, Computing
and Communications Division
Information Directorate

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
SEPTEMBER 2015	FINAL TECHNICAL REPORT	OCT 2013 – MAY 2015

4. TITLE AND SUBTITLE

HIGH SPEED OBLIVIOUS RANDOM ACCESS MEMORY (HS-ORAM)

5a. CONTRACT NUMBER
FA8750-14-C-0012

5b. GRANT NUMBER
N/A

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Radu Sion

5d. PROJECT NUMBER
HSOR

5e. TASK NUMBER
PM

5f. WORK UNIT NUMBER
I1

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Private Machines Inc.
164 20th Street #3D
Brooklyn, NY 11232

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Air Force Research Laboratory/RITA
525 Brooks Road
Rome NY 13441-4505

10. SPONSOR/MONITOR'S ACRONYM(S)
AFRL/RI

11. SPONSOR/MONITOR'S REPORT NUMBER
AFRL-RI-RS-TR-2015-205

12. DISTRIBUTION AVAILABILITY STATEMENT

Approved for Public Release; Distribution Unlimited. DARPA DISTAR CASE # 07-022
Date Cleared: JUL 27, 2015

13. SUPPLEMENTARY NOTES

14. ABSTRACT

High-Speed Oblivious RAM (HS-ORAM) started with a simple thesis: hardware roots of trust can be feasibly deployed as root of trust anchors in the design of secure outsourced data access protocols. HS-ORAM deploys a number of server-side software components running inside tamper-proof secure coprocessors (SCPUs). Employing Oblivious RAM techniques prevents the server from gaining knowledge about the transactions that are occurring within the SCPU from the access patterns. However, traditional ORAM techniques incur expensive overhead penalties. Our approach is to break the ORAM design into a secure composition of multiple smaller ORAMs, thus increasing the throughput of the ORAM mechanism. This report shows that SCPUs can be successfully deployed as anchors of trust in systems enforcing security properties such as access privacy. Results suggest significant improvements of up to 2 orders of magnitude over existing work, especially for large data sets, complex queries, and scenarios requiring the enforcement of query- and content-based query policies beyond simple access control.

15. SUBJECT TERMS

Oblivious Random Access Memory, Hardware-based Security, Embedded Hardware Roots of Trust, Partitioned ORAM, Trusted Hardware

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			**JONATHAN HEINER**
U	U	U	UU	37	19b. TELEPHONE NUMBER (*Include area code*) **315-330-7750**

TABLE OF CONTENTS

Section **Page**

LIST OF FIGURES

Figure **Page**

LIST OF TABLES

Table **Page**

PREFACE

This document represents the final report and guiding roadmap for the High Speed Oblivious Random Access Memory (HS-ORAM) project.

ACKNOWLEDGEMENTS

This work would not have been possible without the excellent support received from Mark Heiligman, Bill Paulsen, Konrad Vesey, and all the other members of the Intelligence Advanced Research Project Activity (IARPA) team, and Jonathan E. Heiner from the Air Force Research Laboratory in Rome, NY.

1.0 SUMMARY

High-Speed Oblivious RAM (HS-ORAM) started with a simple thesis: hardware roots of trust can be feasibly deployed as root of trust anchors in the design of secure outsourced data access protocols. This document describes designs and associated benchmarking results that validate the HS-ORAM thesis.

2.0 INTRODUCTION

When querying data placed remotely, significant challenges face security-conscious clients manipulating sensitive data. Practical assurances of privacy, confidentiality and correctness are essential, especially in intelligence and government frameworks.

Existing research addresses outsourcing security aspects, including access privacy, searches on encrypted data, range queries, and aggregate queries. In many of these works the use of cryptography limits the types of operations that can be performed, leading to fundamental expressiveness and practicality constraints.

Recent theoretical cryptography results provide hope by proving the existence of universal homomorphisms, i.e., encryption mechanisms that allow computation of arbitrary functions without decrypting the inputs [12]. Unfortunately practical instances of such mechanisms seem to be decades away from being truly deployment-ready [5].

Ideas have also been proposed to leverage tamper-proof hardware to privately process data server-side, ranging from smartcard deployment [9] in healthcare, to more general database operations [1, 8, 11].

However, traditional wisdom suggests that deploying trusted hardware is generally feasible only for specialized operations in niche-markets such as banking and Automated Teller Machines (ATM) security while generally impractical due to performance limitations and high acquisition costs. This idea has propagated also to the secure outsourcing realm mainly due to the belief that practical security against curious and malicious insiders can be achieved in "software" only, e.g., by wisely applied combinations of cryptography and data security. Further, this has been confirmed for *correctness* assurances (no privacy) where cryptography has been applied for simple queries such as search and range queries [6, 7, 10].

Yet, as soon as data confidentiality is of concern, data will need to be encrypted before outsourcing. Once encrypted, solutions can be envisioned that: (A) transfer data client-side where it can be decrypted and queried, (B) deploy cryptographic constructs server-side to process encrypted data directly or indirectly, and (C) process the encrypted data server-side inside tamper-proof enclosures of trusted hardware (which the clients trust). In view of novel results [2–4] it is now possible to quantify and compare the costs of each of these cases. In outsourced contexts, computation inside secure hardware processors (C) turns out to be orders of magnitude cheaper than any equivalent software cryptography (B). Similarly, transferring (even a small subset of) the data to the client (for decryption and processing) in (A) is significantly more expensive than (C). The intuition behind this has to do with computing primitives in both trusted and common hardware as well as the cost and overheads of networking. The cost of a Central Processing Unit (CPU) cycle in trusted hardware (56+ US picocents, 1 picocent = 10^{-14} USD) is several *orders* of magnitude cheaper than any equivalent existing (or future) cryptography even for simple addition operations – requiring large numbers of cycles.

These recent insights [4] into the cost-performance trade-off can be used to show that, at scale, in outsourced contexts, computation inside secure processors is orders of magnitude cheaper than any equivalent software cryptography.

Simply put, it is significantly cheaper (by factors of up to 10,000) to process data privately inside relatively expensive server-hosted trusted tamper-proof processors in close data proximity than to perform the expensive cryptography needed to process the data encrypted on plain server

hardware with privacy. Moreover, while cryptography allows only a limited set of operations, trusted hardware-based designs allow for arbitrary expressiveness.

We refer the reader to the HS-ORAM Hardware Capabilities Profile (FA8750-14-C-0012 CLIN 0002 / Item A003 / 0001AA) for a more in-depth discussion.

3.0 METHODS, ASSUMPTIONS, AND PROCEDURES

3.1 Deployment Overview

HS-ORAM (Figure 1) deploys a number of server-side software components running inside tamper-proof secure coprocessors (SCPUs).

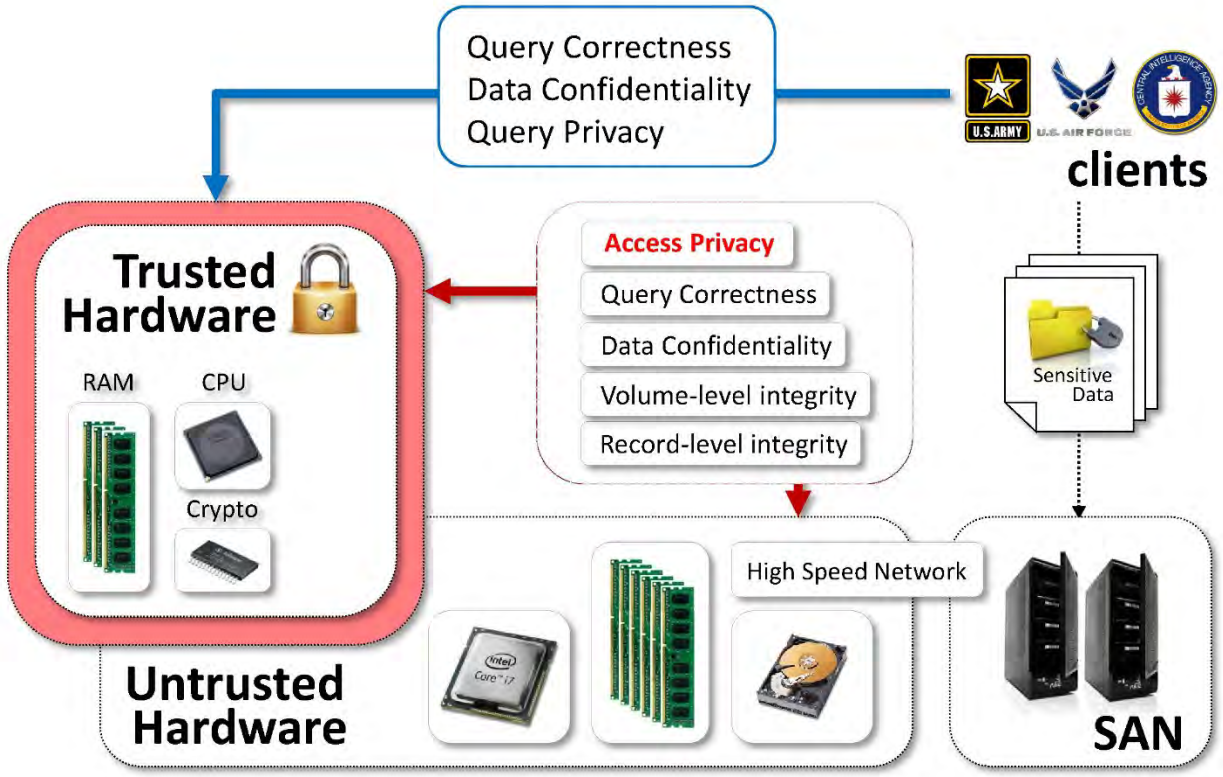

Figure 1. HS-ORAM Deployment Overview

To better structure the discourse and understand exact properties and limitations of different secure data processing designs it is important to first define and structure what is desired, even at an early-stage overview-level.

There exists a large set of security and functionality properties to consider. We are clustering them into the following dimensions: (1) functionality, (2) security properties, (3) parties trust assumptions, (4) technical assumptions, (5) return on investment (ROI), and (6) scale requirements.

Table 1. HS-ORAM handles a certain set of security and functionality dimensions out of a large multi-dimensional space.

Functionality Dimension	
(a) simple block retrieval	✓
(b) file system + namespaces	
(c) simple keyword remote search	✓
(d) boolean search	✓
(e) simple relational algebra	
(f) arbitrary relational algebra	
(g) data-centric policy enforcement	
(h) custom processing modules	✓
(i) recoverability (journaling)	
(j) shareability	
Security Properties Dimension	
(a) data confidentiality	✓
(b) data integrity	
(c) dataset integrity (!)	
(d) fork consistency	
(e) access privacy	✓
(f) remote attestation	✓
(g) access control	
(h) Multi-party computation (MPC)	
(i) Multi-Level Security (MLS) policies	
(j) custom policies	

(k) history independence	
Assumptions: Parties Trust Dimension	
(a) server: just curious	✓
(b) server: trusted to enforce access control	
(c) server: actively malicious	
(d) client: colluding with server	
Assumptions: Technical Dimension	
(a) Advanced Encryption Standard (AES)	✓
(b) RSA	✓
(c) Random Number Generator (RNG)	✓
(d) tamper resistance	✓
(e) no side-channels	✓
(f) fast zeroization	✓
(g) cost of attack	✓
ROI (Performance/$) Dimension	
(a) simple metric: $/transaction.	
(b) complex metrics: $/unit of transaction complexity.	
(c) Monetary benefits of system scalability (i.e., scale out vs. scale up; is not hyper-optimized but can scale up easily and linearly)	
(d) Monetary benefits of design extensibility (i.e., query-specific hyper-optimized implementation vs. extensible designs)	

Scale Requirements	
(a) Target Data Set Sizes	1TB+
(b) Target Performance	100qpm+

3.2 Architecture

Section 3.2 discusses the main architectural components of HS-ORAM and its evolution. Section 3.3 provides a number of implementation details. Section 4.0 elaborates on the experimental results yielded in benchmarking the developed HS-ORAM components.

3.2.1 Overview.

Data Pumps. In data-centric processing, software-hardware complexes can be viewed effectively as "data pumps" operating at certain internal throughput – limited by internal component (CPU, hardware engines etc) speeds. Further, these data pumps are endowed with a number of input and output conduits, each inherently rate-limited by their (mostly hardware) capabilities.

An ideal design leverages the available types and number of components of different "pumping" capabilities to result in a high-utilization operation mode that maximizes the overall throughput while minimizing per-transaction costs.

SCPU. The SCPU's internal pump is a PowerPC 460EX, running at 600Mhz in the production version (theoretical frequencies range up to 1GHz, but overheating is an issue). Its input/output (I/O) conduits include two Ethernet ports (Marvell 88E1121R Gbps transceiver, 1Gbps theoretical throughput) and a Serial Advanced Technology Attachment (SATA) I/O port (460EX PPC chipset, 3Gbps theoretical throughput).

This is further exacerbated by the fact that the conduits require kernel cycles at different degrees. For example, in the absolute best scenario, the maximum observed Ethernet throughput does not exceed 500Mbps, achieved at 97% CPU utilization!

Networked Internet Small Computer System Interface (iSCSI) volumes deploying customized PowerPC iSCSI logic, run at no more than 25.6MB/s (204.8Mbps). The SATA disk interface achieves no more than 937Mbps raw data throughput at 78% CPU utilization (or 621Mbps at 56% CPU utilization measured with a different tool – hdperf) using a Patriot Pyro SE PPSE120GS25SSDR 2.5" 120GB Solid State Drive (SSD), with a 3.13.7 customized stripped down Linux kernel with SATA support.

These data points are illustrated in Figure 2.

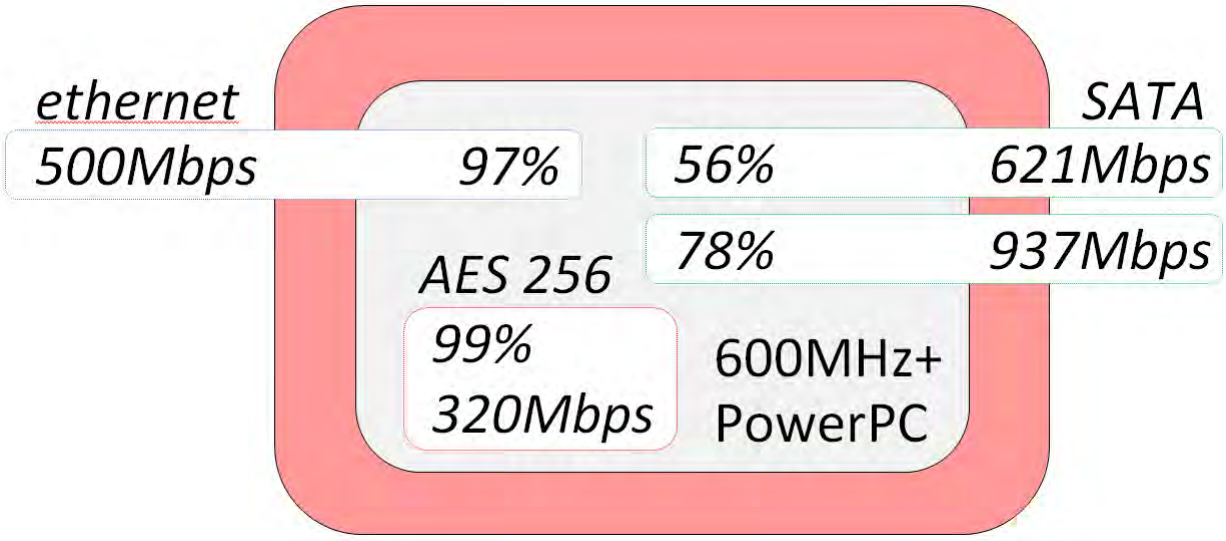

Figure 2. The SCPU as a data pump.

In any design, the SCPU needs access to (i) a primary data store and (ii) the ability to communicate with other parties (other SCPUs and remote clients). Before deciding on any higher-level protocol properties and details, the first choice that needs to be made is how to best maximize the utilization for the SCPU pump.

Considering the numbers above, it is clear that optimality can only be achieved by deploying the SATA interface fully in the data path – since it is a conduit that provides high throughput at lower CPU utilization than Ethernet. This immediately directs the overall design towards a distributed setup in which high-capacity SATA 1TB+ disks – or SSDs if needed to saturate the internal CPU pump and the protocol level logic does not achieve high CPU utilization – are connected to individual SCPUs on their SATA bus.

Since the disks are not contained within the trusted enclosure of the SCPU, it is important that they are accessed with access privacy assurances using a base Oblivious Random Access Memory (ORAM) mechanism. We chose the currently fastest ORAM mechanism with client storage requirements that can be accommodated without recursion within the 512MB of Dynamic Random Access Memory (DRAM) available, Path ORAM [14]. We discuss Path ORAM below.

Further, since we require data confidentiality, and encryption is performed in software, the SCPU's overall "pumping" throughput is still dominated by its internal CPU. This is so because its internal 33MHz crypto engine (MAXQ 1103) processor is simply too slow to place on the data path.

The CPU can run symmetric key crypto at between 150 - 320Mbps at 99.9% CPU utilization. For example we have benchmarked Advanced Encryption Standard (AES) 256 Cipher Block Chaining (CBC) at 176Mbps using OpenSSL 0.9.8g. An optimized AES 256 Counter Mode (CTR) version we custom-wrote for the SCPU peaked at 320Mbps (99% CPU utilization).

Finally, to achieve a target scale we need to understand how to combine a number of SCPUs securely (given our data confidentiality and access privacy policy) and most importantly efficiently.

There are several meta-design choices with different pros and cons elements. Within the space and time constraints of the current seedling however, we choose an existing protocol, with strong security proofs and community vetting.

The "Partitioned ORAM" [13], see Figure 3, allows the combination of a number of individual ORAMs into a larger ORAM structure with full privacy guarantees. We discuss the Partitioned ORAM approach in the following.

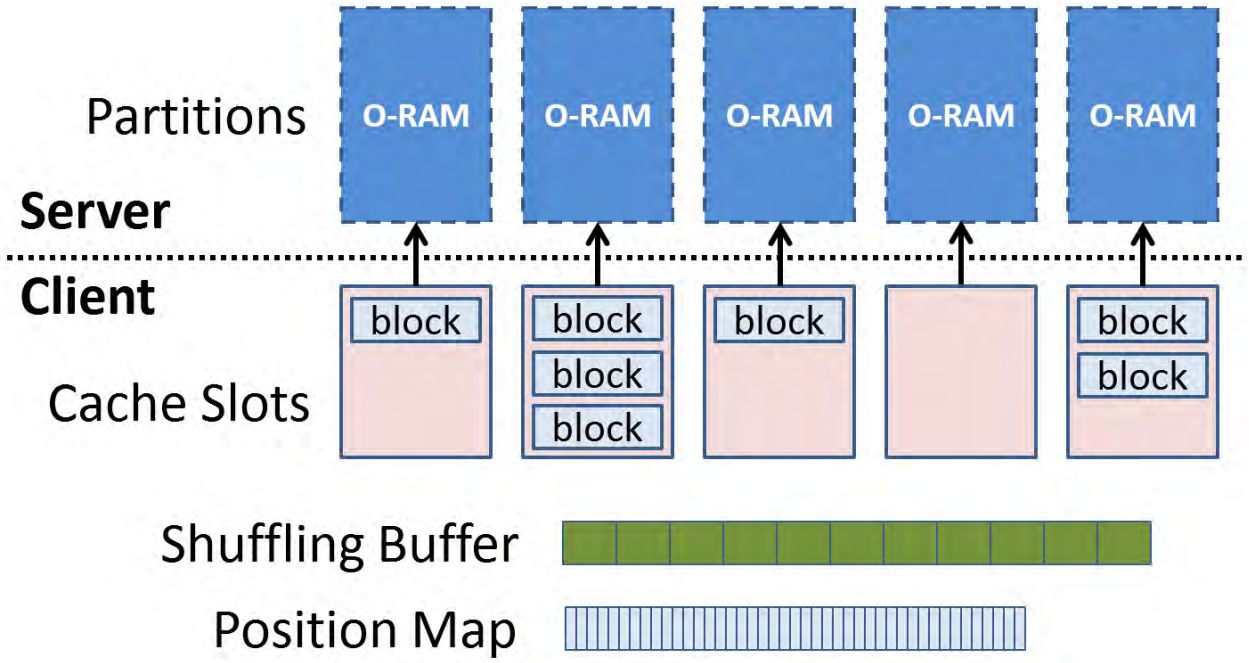

Figure 3. Partitioned ORAM Overview [13]

3.2.2 Partitioned ORAM.

The main idea of [13] is to "partition a bigger O-RAM into smaller O-RAMs, and employ a background eviction technique to obliviously evict blocks from the client-side cache into a randomly assigned server-side partition."

More specifically, [13] partitions a single O-RAM of size N blocks, into P different ORAMs of size N/P blocks each:

> "This allows us to break down a bigger O-RAM into multiple smaller O-RAMs. The idea of partitioning is motivated by the fact that the major source of overhead in existing O-RAM constructions arises from an expensive remote oblivious sorting protocol performed between the client and the server. Because the oblivious sorting protocol can take up to $O(\sqrt{N})$ time, existing O-RAM schemes require $\Omega(N)$ time in the worst-case or

> have unreasonable $O(\sqrt{N})$ amortized cost. We partition the Oblivious RAM into roughly $P = \sqrt{N}$ partitions, each having $\sqrt{N}$ blocks approximately. This way, the client can use N blocks of storage to sort/reshuffle the data blocks locally, and then simply transfer the reshuffled data blocks to the server. This not only circumvents the need for the expensive oblivious sorting protocol, but also allows us to achieve $O(\sqrt{N})$ worst-case cost. Furthermore, by allowing reshuffling to happen concurrently with reads, we can further reduce the worst-case cost of the practical construction to *O(log(N))*. While the idea of partitioning is attractive, it also brings along an important challenge in terms of security. Partitioning creates an extra channel through which the data access pattern can potentially be inferred by observing the sequence of partitions accessed. Therefore, we must take care to ensure that the sequence of partitions accessed does not leak information about the identities of blocks being accessed. Specifically, our construction ensures that the sequence of partitions accessed appears pseudo-random to an untrusted server."

For more details and security proofs, we direct the reader to [13].

In HS-ORAM, the initial reason for introducing the partitioning approach had to do with mitigating client-side storage limitations – effectively allowing the client to shuffle each individual's ORAM levels efficiently, client-side.

> "The idea of partitioning is motivated by the fact that the major source of overhead in existing O-RAM constructions arises from an expensive remote oblivious sorting protocol performed between the client and the server." [13]

In the case of HS-ORAM we are using the partitioning approach for a slightly different purpose: secure composition of multiple smaller ORAMs. A special HS-ORAM Controller (Figure 1 and Figure 4) runs the partitioning logic and controls a number of individual SCPU ORAMs (running Path ORAM [14]). Each individual segment ORAM's reshuffling is done inside each SCPU ORAM and thus the controller does not require a shuffling buffer anymore. Instead, it deploys the segmentation technique simply to combine these multiple ORAMs and achieve a unified larger address space.

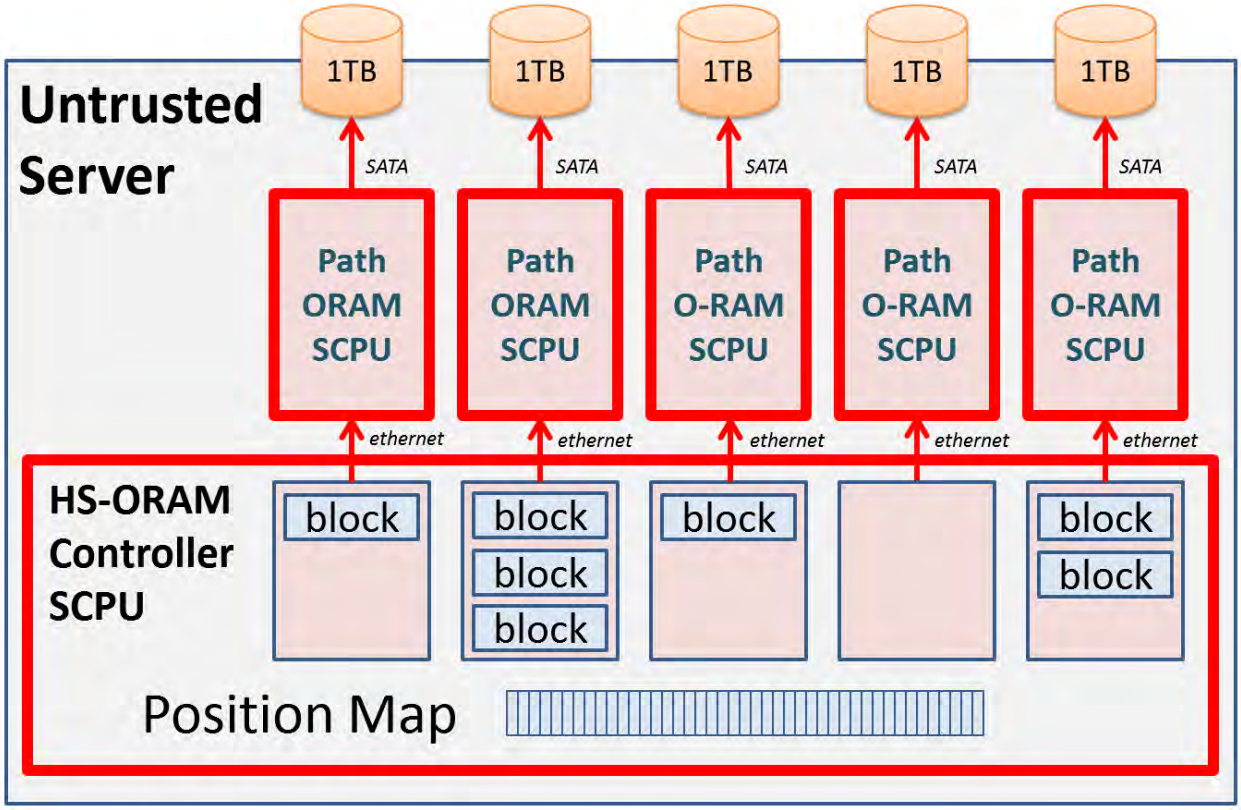

Figure 4. HS-ORAM deploys a partitioning approach to securely compose a number of customized individual Path ORAM instances running inside SCPUs. The partitioning is handled by a SCPU-hosted HS-ORAM Controller.

3.2.3 Path ORAM.

Path ORAM [14] and its variants are currently the fastest ORAM mechanism with the lowest client-side CPU load. And, despite its seemingly large O(n) client-side storage requirement – which seems to defeat the raison d'etre of the outsourcing premise in the first place – the constants are small. We have determined that with certain optimizations, we can accommodate its storage footprint without a recursive construction within the 512MB of DRAM available.

Path ORAM [14] is built around a tree of "buckets", each bucket containing a number of data blocks. A data block is randomly (re)associated with a specific leaf (after any operation targeting the block). The association is maintained in a client-hosted position map. Data blocks are stored in any bucket residing on the path from the block's associated leaf to the root of the tree. After operations, blocks are kept client-side in a "stash" until they can be evicted to their appropriate location in the tree. Eviction happens during another operation or periodically (Figure 5). More specifically [14]:

> "The client stores a small amount of local data in a stash. The server-side storage is treated as a binary tree where each node is a bucket that can hold up to a fixed number of blocks.
>
> **Main invariant.** We maintain the invariant that at any time, each block is mapped to a uniformly random leaf bucket in the tree, and unstashed blocks are always placed in some

bucket along the path to the mapped leaf. Whenever a block is read from the server, the entire path to the mapped leaf is read into the stash, the requested block is remapped to another leaf, and then the path that was just read is written back to the server.

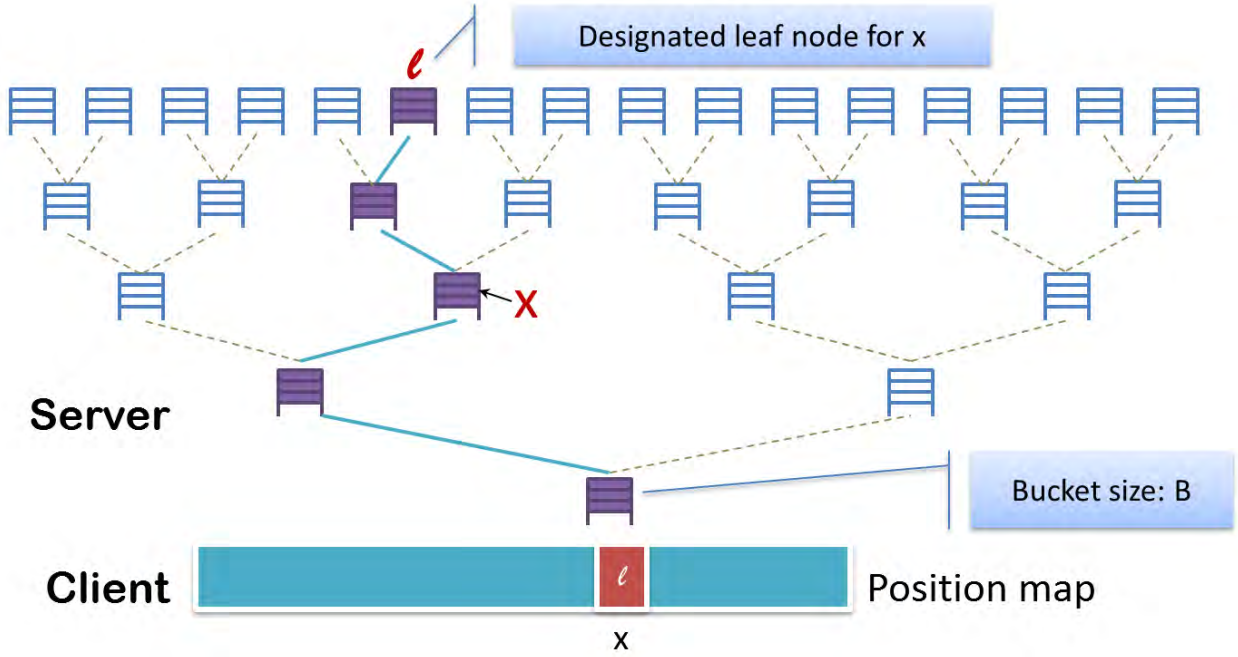

Figure 5. Path ORAM [14] is built around a tree of "buckets" containing data blocks. Data blocks are: (i) randomly (re)associated with specific leafs in in a client-hosted position map; (ii) stored anywhere on the path from "their" leaf to the root of the tree; (iii) after operations, blocks are kept in a client-side "stash" until evicted to their appropriate tree location.

When the path is written back to the server, additional blocks in the stash may be evicted into the path as long as the invariant is preserved and there is remaining space in the buckets."

On the server side [14]:

"data is stored in a tree consisting of buckets as nodes. The tree does not have to necessarily be a binary tree, but we use a binary tree in our description for simplicity.

Binary tree. The server stores a binary tree data structure of height $L = log_2 (N) - 1$ and $2L$ leafs. The tree can easily be laid out as a flat array when stored on disk. The levels of the tree are numbered 0 to L where level 0 denotes the root of the tree and level L denotes the leafs.

Bucket. Each node in the tree is called a bucket. Each bucket can contain up to Z real blocks. If a bucket has less than Z real blocks, it is padded with dummy blocks to always be of size Z. It suffices to choose the bucket size Z to be a small constant such as $Z = 4$ (see Section 5.1 [in [14]]).

Path. Let $x \in \{0, 1, \ldots, 2^L-1\}$ denote the x-th leaf node in the tree. Any leaf node x defines a unique path from leaf x to the root of the tree. We use $P(x)$ to denote set of buckets along the path from leaf x to the root. […]

Server storage size. Since there are about N buckets in the tree, the total server storage used is about $Z \times N$ blocks."

Client-side storage consists of "2 data structures, a stash and a position map":

"**Stash.** During the course of the algorithm, the client locally stores a small number of blocks in a local data structure S called the stash. In Section 6 [of [14]], we prove that the stash has a worst-case size of $O(\log N) \times \omega(1)$ blocks with high probability. In fact, in Section 5.2 [of [14]], we show that the stash is usually empty after each ORAM read/write operation completes.

Position map. The client stores a position map, such that $x := position[a]$ means that block a is currently mapped to the x-th leaf node this means that block a resides in some bucket in path $P(x)$, or in the stash. The position map changes over time as blocks are accessed and remapped.

Bandwidth. For each load or store operation, the client reads a path of $Z \log(N)$ blocks from the server and then writes them back, resulting in a total of $2Z \log(N)$ blocks bandwidth used per access. Since Z is a constant, the bandwidth usage is $O(\log(N))$ blocks."

Path ORAM read/write operations pre serve two invariants: (i) "at any time, each block is mapped to a uniformly random leaf bucket in the tree" and (ii) "unstashed blocks are always placed in some bucket along the path to the mapped leaf".

"The Access protocol can be summarized in 4 simple steps:

1. Remap block: Randomly remap the position of block a to a new random position. Let x denote the blocks old position.

2. Read path: Read the path $P(x)$ containing block a.

3. Update block: If the access is a write, update the data stored for block a.

4. Write path: Write the path back and possibly include some additional blocks from the stash if they can be placed into the path. Buckets are greedily filled with blocks in the stash in the order of leaf to root, ensuring that blocks get pushed as deep down into the tree as possible. A block a' can be placed in the bucket at level l only if the path $P(position[a'])$ to the leaf of block a' intersects the path accessed $P(x)$ at level l. In other words, if $P(x, l) = P(position[a'], l)$."

For more details and security proofs, we direct the reader to [14]

In HS-ORAM, Path ORAM is deployed in HS-ORAM as the individual segment building block driven by a partitioned ORAM running in a controller node. With the 512MB SCPU DRAM individual Path ORAM instances can control up to 4TB.

3.2.4 Boolean Search.

The HS-ORAM seedling also implemented a boolean keyword search capability built using the components outlined above. Remote clients are able to submit boolean keyword search queries against all the blocks in the data set with computational access privacy and data confidentiality.

We are considering the ability to handle up to one million distinct keywords, and data sets up to 1TB of data (up to one billion data blocks).

The architecture includes an indexing ORAM ("B-IDX:ORAM"), a data-hosting ORAM ("DATA:ORAM"), and a dictionary ("KW:RAM") hosted in the DRAM of the "Search Controller" (Figure 6).

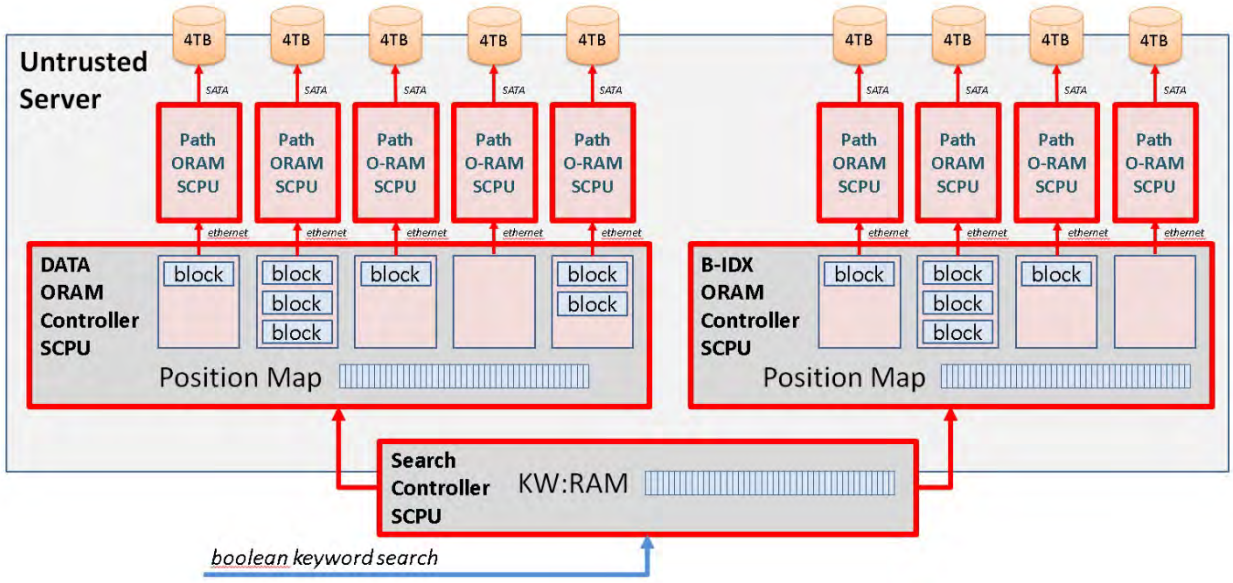

Figure 6. HS-ORAM Boolean Search Architecture

B-IDX:ORAM. The index maps search keywords – or, more precisely, keyword identifiers (IDs) – to their corresponding posting lists, the set of block IDs that contain the corresponding keyword. Keywords and data blocks are uniquely identified using 32 bit IDs. For the considered data points, we estimate the entire index to grow up to 1TB.

DATA:ORAM. The data-hosting ORAM stores actual data blocks. It allows oblivious block retrieval using a 32 bit block ID as a reference. We expect this ORAM to grow beyond 20TB and store up to 4TB of actual data.

KW:RAM. The dictionary maps keywords to 32 bit keyword IDs. For the considered scale, this dictionary does not exceed 60MB and can be stored in the DRAM of the Search Controller for efficiency.

Search Controller. The index and data ORAMs are managed by the Search Controller SCPU which also hosts the dictionary securely in its internal DRAM. The Search Controller operates as follows:

1. Receive search query from remote client.

2. Map corresponding keywords to keyword IDs using the dictionary.

3. Dispatch per-keyword queries to the B-IDX:ORAM to retrieve corresponding posting lists.

4. Perform boolean query locally securely.

5. Dispatch per-block queries to the DATA:ORAM to retrieve the corresponding data blocks.

6. Return blocks together with signature back to remote client.

3.3 Implementation Details

In this section we present validating benchmarking results of the above component

HS-ORAM has been implemented in C for efficiency.

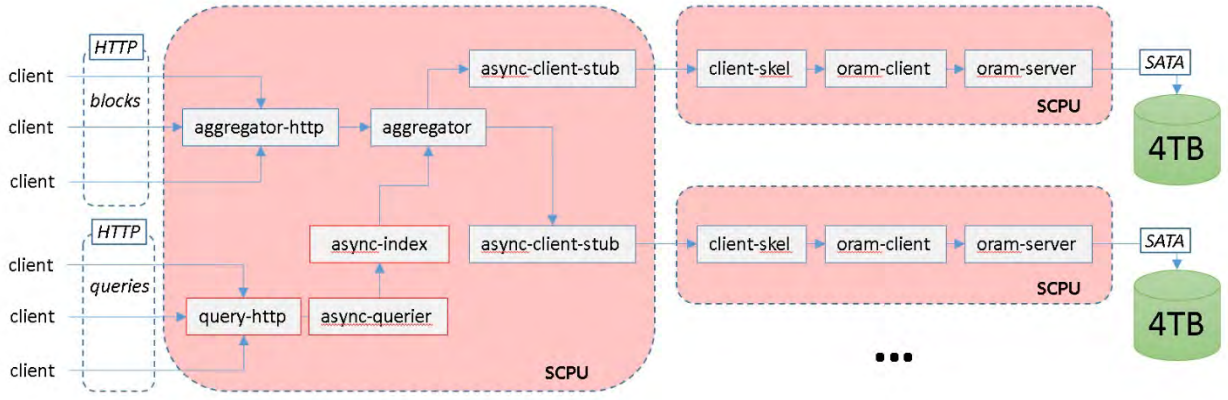

Figure 7. Maximal asynchronicity (for scale-out) HS-ORAM source code invocation data path. Remote HTTP clients can access either data blocks directly or submit boolean keyword queries. Each rectangle corresponds to a source code module. In progress work includes the async-inverted-index.c module and the index-http module.

The HS-ORAM index is designed to allow full data path parallelism and to benefit from a scale-out nature in which multiple SCPUs are working together. This is achieved by providing an end-to-end non-blocking, fully asynchronous callback data processing paradigm.

Figure 7 illustrates two common data paths in the current implementation. To achieve maximal asynchronicity (for scale-out) in the HS-ORAM invocation data path numerous queueing and synchronization components have been designed. Remote Hypertext Transfer Protocol (HTTP) clients can access either data blocks directly or submit boolean keyword queries. Each rectangle corresponds to a source code module.

The HS-ORAM partitioning logic (aggregator), whose main data processing components are illustrated in Figure 8 already operates asynchronously.

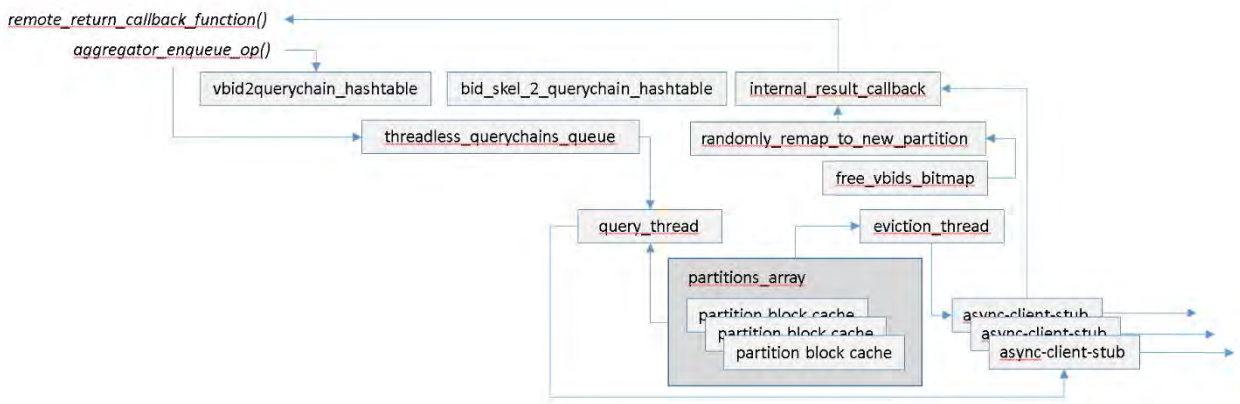

Figure 8. The HS-ORAM aggregator module has the ability to combine a number of remote ORAM servers asynchronously. Selected key components are illustrated.

Further, Figure 8 details some of the key components of the multi-segment aggregator module (aka. "Controller") design. The HS-ORAM aggregator is endowed with the ability to combine a number of remote ORAM servers asynchronously.

Finally, Figure 9 details some of the key components of the asynchronous index module design. The asynchronous index design requires extreme care to achieve full parallelism, isolation and inter-thread synchronization while also preserving the privacy guarantees of the underlying aggregator.

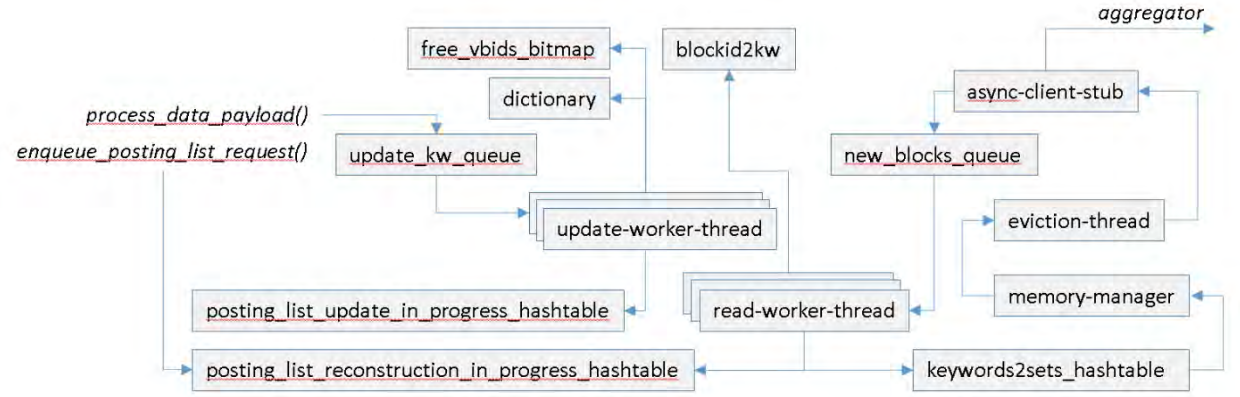

Figure 9. The asynchronous index design requires extreme care to achieve full parallelism, isolation and inter-thread synchronization while also preserving the privacy guarantees of the underlying aggregator. Selected key components of the asynchronous index module are illustrated.

4.0 RESULTS AND DISCUSSION

In this section we present validating benchmarking results of the above components.

4.1 Synchronous single-SCPU Baseline

The synchronous, single-SCPU implementation serves as a baseline of benchmarking the capabilities of a single SCPU ORAM in a single-client scenario. The results are extremely encouraging.

Figure 10 illustrates the blocks/min raw throughput of a single SCPU-based block retrieval interface for different-sized databases in a constrained remote client network setup (3ms latencies, 10Mbps bandwidth).

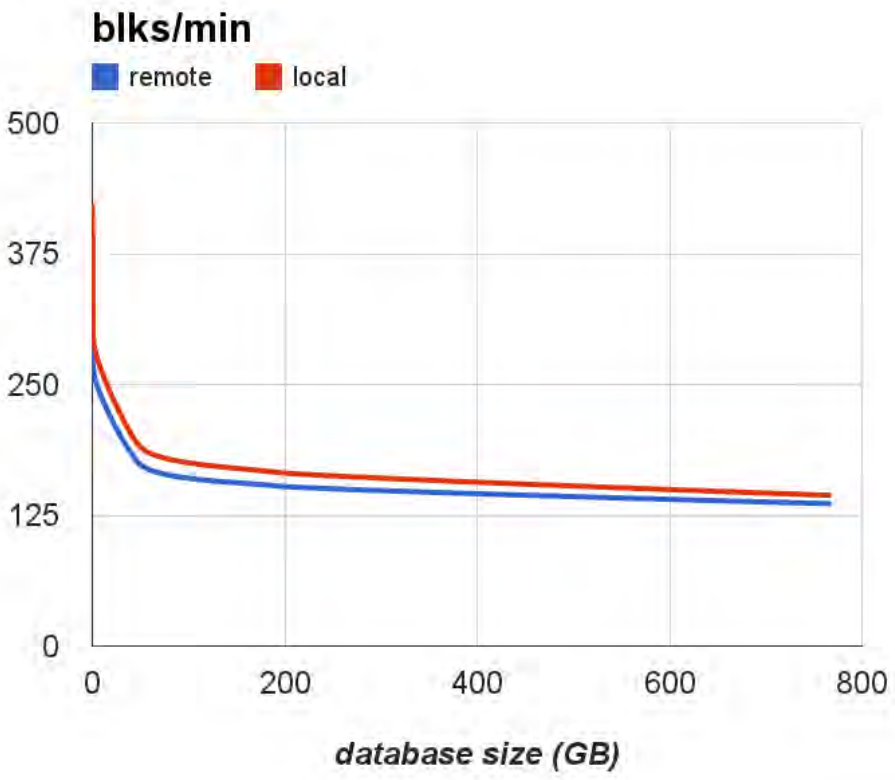

Figure 10. Blocks/min throughput of a single SCPU-based block retrieval interface for different-sized databases in a constrained remote client network setup (3ms latencies, 10Mbps bandwidth). Throughput decreases with increasing database size. "Local" throughput illustrates an intra-SCPU viewpoint (client running inside the same SCPU).

As expected, throughput decreases with increasing database size. "Local" throughput illustrates an intra-SCPU viewpoint (client running inside the same SCPU).

Similarly, Figure 11 illustrates raw Kbps throughput of a single SCPU-based block retrieval interface for different-sized databases in a constrained remote client network setup (3ms latencies, 10Mbps bandwidth). As expected, throughput decreases with increasing database size. "Local" throughput illustrates an intra-SCPU viewpoint (client running inside the same SCPU).

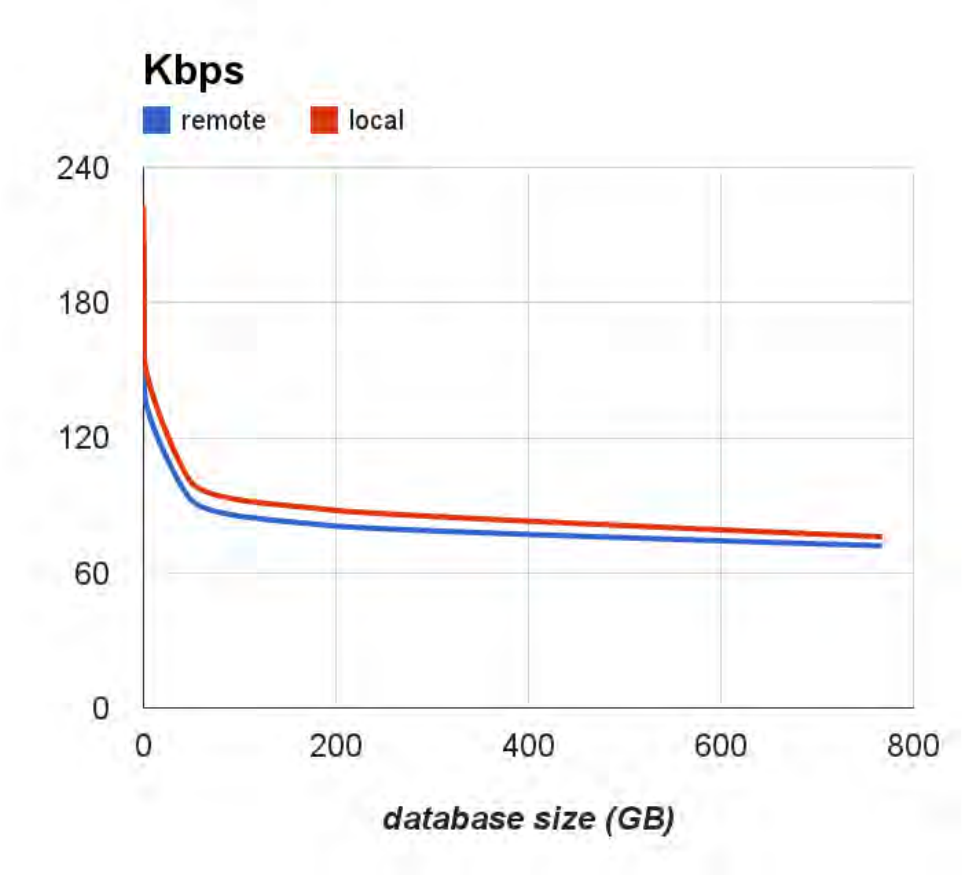

Figure 11. Kbps throughput of a single SCPU-based block retrieval interface for different-sized databases in a constrained remote client network setup (3ms latencies, 10Mbps bandwidth). Throughput decreases with increasing database size. "Local" throughput illustrates an intra-SCPU viewpoint (client running inside the same SCPU).

To understand the impact of block size choices on overall behavior, we also benchmarked the raw throughputs for increasing block sizes, for a fixed database size of 1TB.

Figure 12 illustrates the Kbps throughput of a single SCPU-based block retrieval interface for databases with different block sizes in a constrained remote client network setup (3ms latencies, 10Mbps bandwidth).

As can be seen, overall throughput increases with increasing block size. What this suggests is that a choice of block size should be guided by the typical burst-behavior of the workloads. If workloads are composed of mostly bursts of large data blocks, a larger backend ORAM block size may result in better throughput. Alternately, for workloads composed of numerous, small requests, a small block size may be in order to optimize overhead and unnecessary network bandwidth waste.

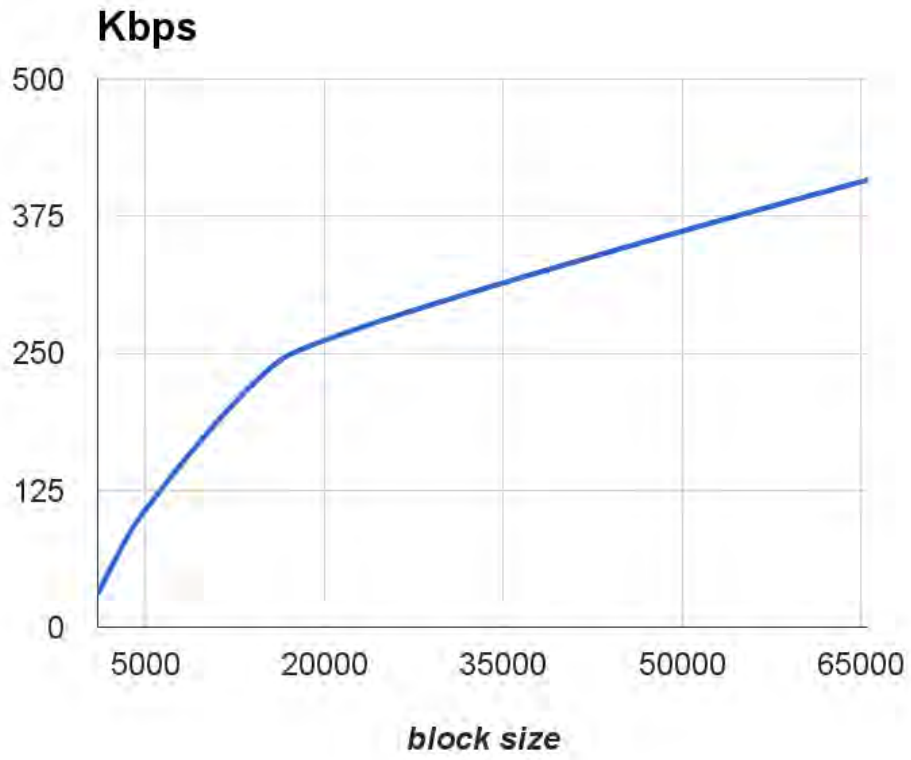

Figure 12. Kbps throughput of a single SCPU-based block retrieval interface for databases with different block sizes in a constrained remote client network setup (3ms latencies, 10Mbps bandwidth). Overall throughput increases with increasing block size.

Similarly, Figure 13 illustrates blocks/min throughput of a single SCPU-based block retrieval interface for databases with different block sizes in a constrained remote client network setup (3ms latencies, 10Mbps bandwidth).

In this case however, throughput decreases with increasing block sizes, since the SCPU processing throughput is fixed. This is expected and in line with the above Kbps result. Together they basically show that overall, while, with increasing block sizes, fewer blocks are being processed per unit of time, since the blocks are larger, overall Kbps throughput increases.

We have implemented a synchronous indexing and multi-keyword boolean query interface on top of the single-SCPU synchronous setup described above. Its performance is described in Figure 14 which illustrates the end-to-end performance of a single SCPU-based boolean keyword search query interface on a 1TB database with 4KB blocks and a small dictionary of 1600 keywords in a constrained remote client network setup (3ms latencies, 10Mbps bandwidth).

As can be seen, throughput decreases with increasing number of query keywords. This is expected, since multiple keywords require multiple index entries' retrieval and associated ORAM processing. The peak query rate of almost 140 queries / minute is much higher than our initial target.

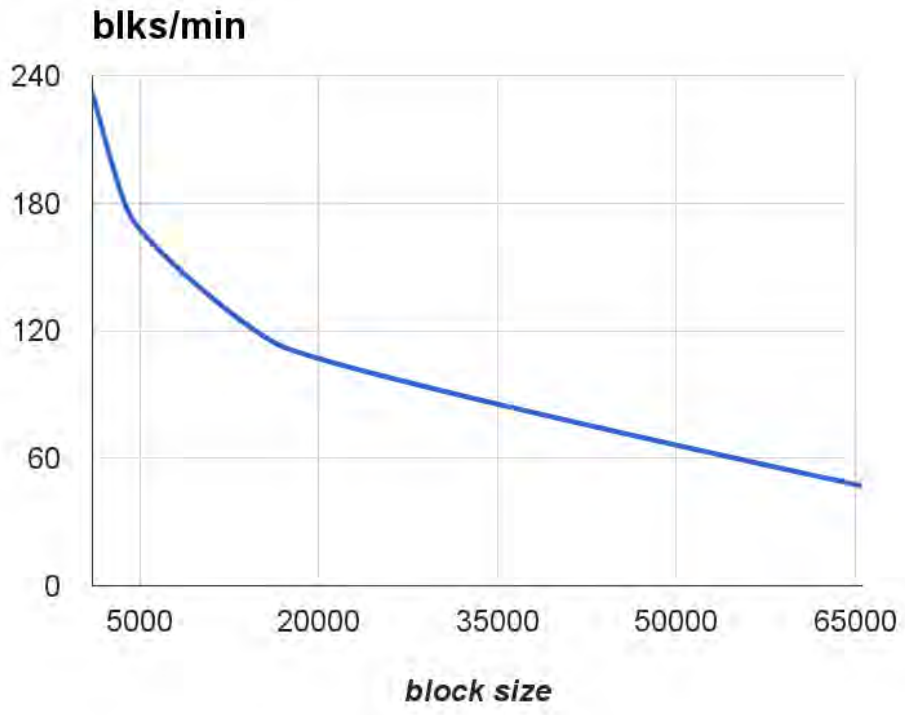

Figure 13. Blocks/min throughput of a single SCPU-based block retrieval interface for databases with different block sizes in a constrained remote client network setup (3ms latencies, 10Mbps bandwidth). Straightforwardly, throughput decreases with increasing block sizes, since the SCPU processing throughput is fixed.

For this result, SCPU utilization is high (65-80%). Thus, since we have managed to saturate the SCPU, as planned, to increase this throughput we will resort to parallelism as discussed above.

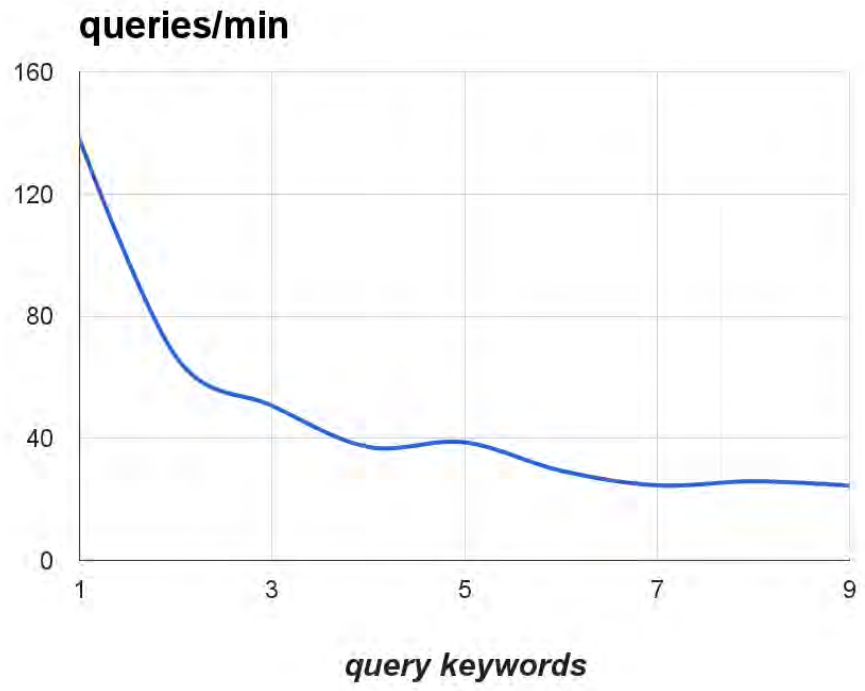

Figure 14. End-to-end performance of a single SCPU-based boolean keyword search query interface on a 1TB database with 4KB blocks and a small dictionary of 1600 keywords in a constrained remote client network setup (3ms latencies, 10Mbps bandwidth). Throughput decreases with increasing number of query keywords.

4.2 Asynchronous multi-SCPU Implementation

High-throughput parallelism can only be achieved using an asynchronous design. This design is outlined in Figure 7, Figure 8, and Figure 9.

Figure 15 illustrates blocks/min throughput of an aggregated, asynchronous, multi SCPU-based block retrieval interface for different sized databases with different block sizes in a constrained remote client network setup (3ms latencies, 10Mbps bandwidth).

As can be seen, similar to the synchronous behavior in Figure 13 straightforwardly, throughput decreases with increasing block sizes. Further, as expected, throughput increases with increasing number of SCPUs. The increase is not linear due to overhead.

The throughput with two SCPUs is about 75% higher than that of a single SCPU. These results seem to suggest that this level of increase continues for additional SCPUs.

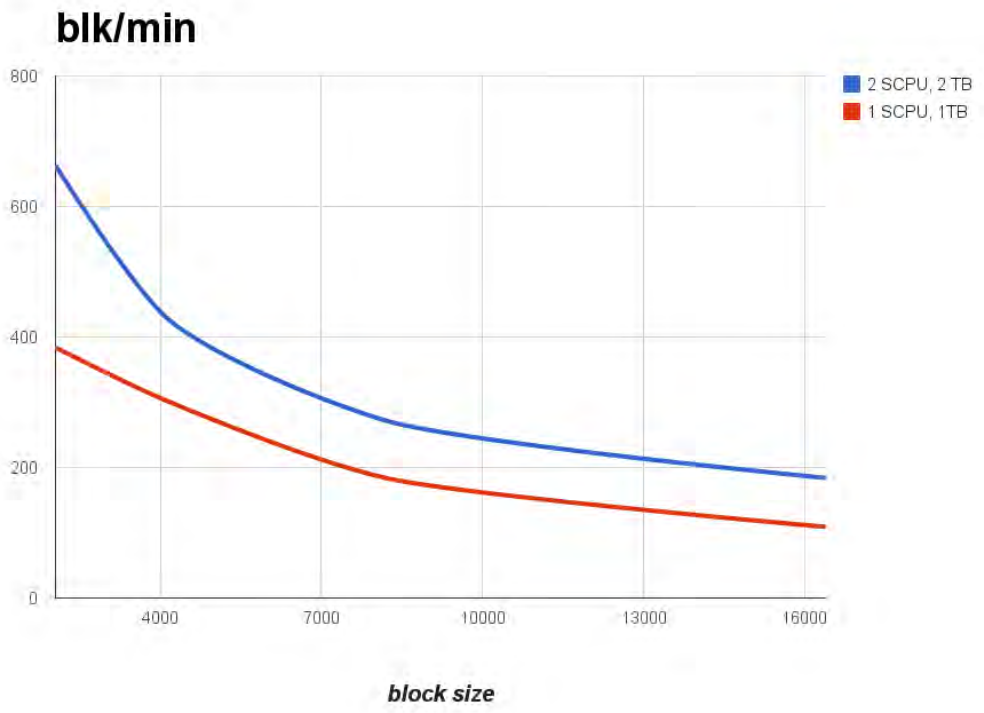

Figure 15. Blocks/min throughput of an aggregated, asynchronous, multi SCPU-based block retrieval interface for different sized databases with different block sizes in a constrained remote client network setup (3ms latencies, 10Mbps bandwidth). Straightforwardly, throughput decreases with increasing block sizes. Further, throughput increases with increasing number of SCPUs.

Similarly, Figure 16 illustrates Kbps throughput of an aggregated, asynchronous, multi SCPU-based block retrieval interface for different sized databases with different block sizes in a constrained remote client network setup (3ms latencies, 10Mbps bandwidth).

As expected, and consistent with the synchronous case results in Figure 12 throughput increases with increasing block sizes.

Further, a scale-up behavior is observed, and throughput increases with increasing number of SCPUs. Interestingly, since each SCPU participates with its own data storage, multiple SCPUs also mean increased capacity. For example, the 2 SCPU ORAM features a 2TB database and an almost 400Kbps throughput for 16KB block sizes.

We have built an asynchronous query processor and indexer on top of the asynchronous block interfaces discussed above. The processor exposes a multi-keyword boolean keyword search.

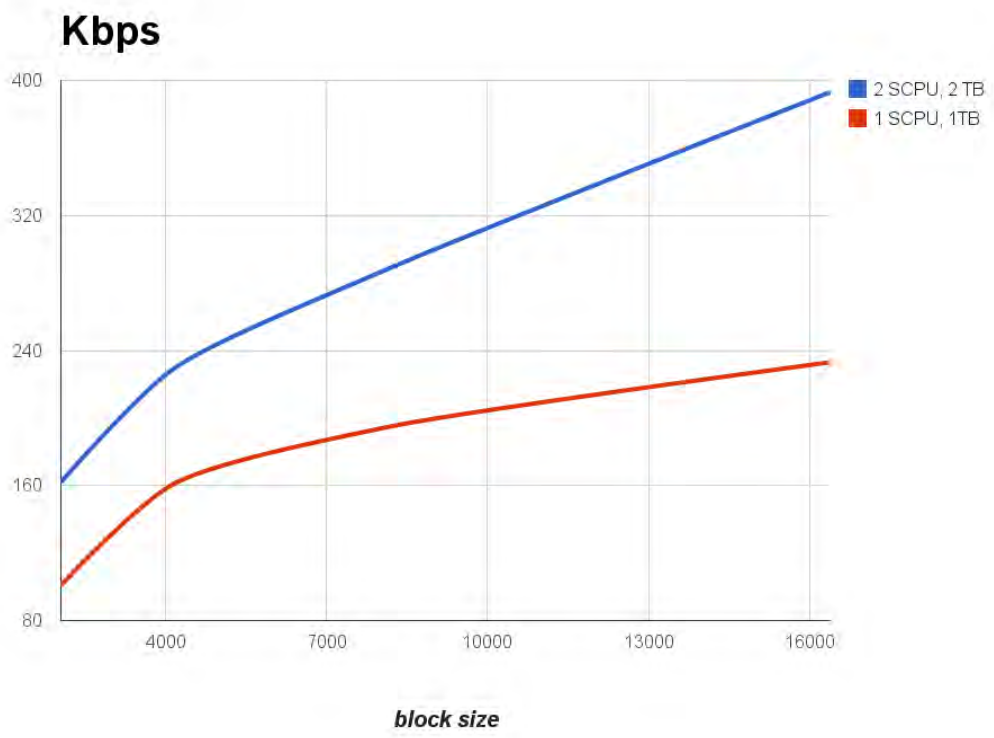

Figure 16. Kbps throughput of an aggregated, asynchronous, multi SCPU-based block retrieval interface for different sized databases with different block sizes in a constrained remote client network setup (3ms latencies, 10Mbps bandwidth). Throughput increases with increasing block sizes. Further, throughput increases with increasing number of SCPUs.

Figure 17 shows the behavior of this interface. Specifically, it illustrates the end-to-end performance of an aggregated, asynchronous, multi SCPU-based boolean keyword search query interface on a database with a small dictionary of 1600 keywords in a constrained remote client network setup (3ms latencies, 10Mbps bandwidth).

As in the case of the synchronous design results illustrated in Figure 14, throughput decreases with increasing number of query keywords.

A good scale-out behavior can also be seen and throughput increases with increasing number of SCPUs. Specifically, the 2 SCPU, 2 TB database is about 38% faster than the 1 SCPU, 1 TB database (4096 byte blocks).

It is important to note that as expected from the synchronous results, throughput increases with decreasing block size (2048 to 1024). For example, the 2 SCPU, 2 TB database setup behaves about 45% faster for block sizes of 1024 bytes when compared with the case of 4096 byte block sizes.

Finally, we note a U-shaped behavior with increasing number of keywords. This is likely caused by aggregator-side caching which securely caches previous results (such as index pages) aggregator-side and reuses them in future queries.

To better understand this behavior, and eliminate other variables, we benchmarked a setup with and without aggregation for a 2 SCPU, 2 TB database scenario with 1024 byte blocks.

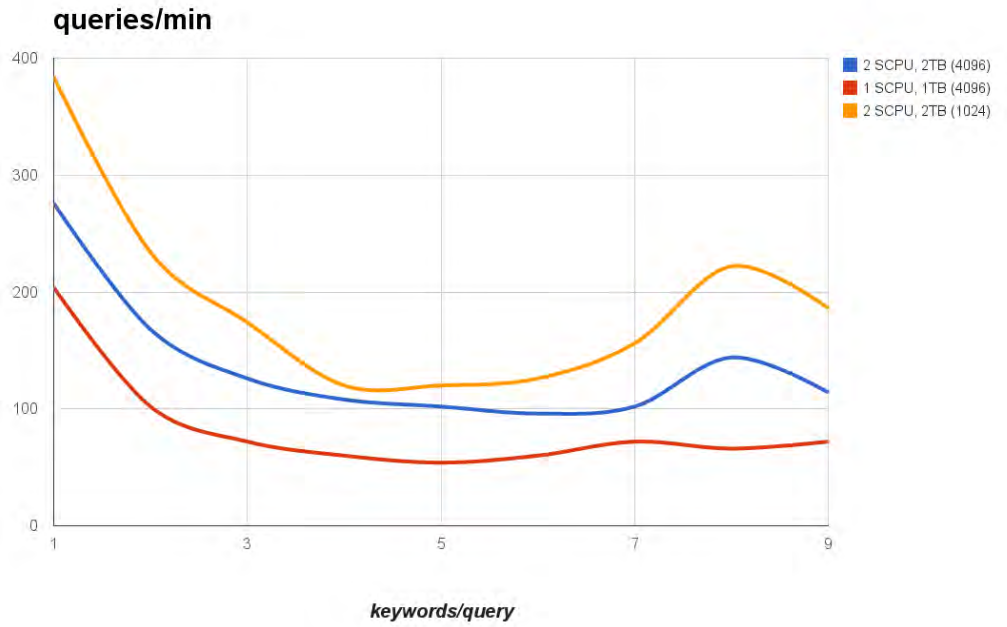

Figure 17. End-to-end performance of an aggregated, asynchronous, multi SCPU-based boolean keyword search query interface on a database with a small dictionary of 1600 keywords in a constrained remote client network setup (3ms latencies, 10Mbps bandwidth). Throughput decreases with increasing number of query keywords. Throughput increases with increasing number of SCPUs. Throughput increases with decreasing block size (2048 to 1024). Aggregator-side caching results in the U-shaped behavior.

The results illustrated in Figure 18 confirm our hypothesis and disabling aggregator-side caching results in a decreasing throughput behavior as expected.

Figure 18 illustrates the end-to-end performance of an aggregated, asynchronous, 2 SCPU-based boolean keyword search query interface on a 2 TB database with 1KB blocks and a small dictionary of 1600 keywords in a constrained remote client network setup (3ms latencies, 10Mbps bandwidth). Throughput decreases with increasing number of query keywords. Disabling aggregator-side caching (red curve) results in decreasing throughput (for higher number of keywords).

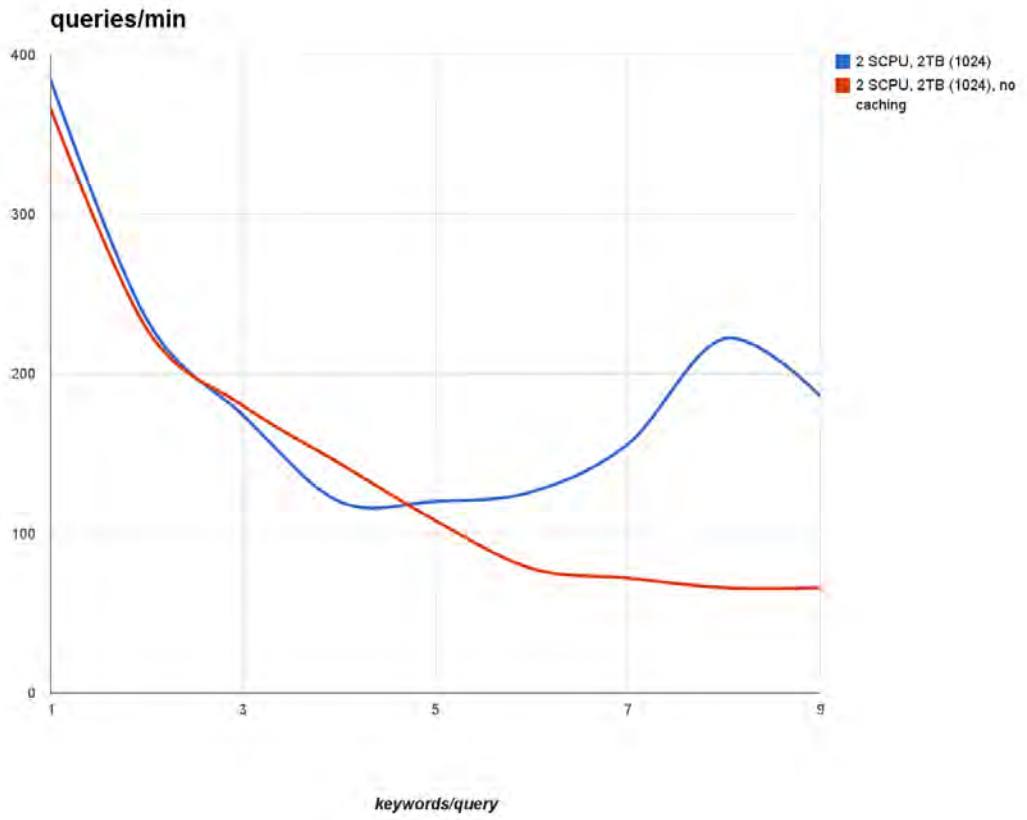

Figure 18. End-to-end performance of an aggregated, asynchronous, 2 SCPU-based boolean keyword search query interface on a 2 TB database with 1KB blocks and a small dictionary of 1600 keywords in a constrained remote client network setup (3ms latencies, 10Mbps bandwidth). Throughput decreases with increasing number of query keywords. Disabling aggregator-side caching (red curve) results in decreasing throughput (for higher number of keywords).

4.2.1 Impact of Rotational Hard Disk Latencies.

While all the results above have been achieved using solid state disks, we also performed experiments to understand the impact of the I/O latencies encountered at the hard disk level.

For data hosted on rotational 7200rpm 1TB hard disks, Figure 19 represents the query/minutes end-to-end performance of an aggregated, asynchronous, 3 SCPU-based Boolean keyword search query interface on databases of different sizes, block sizes, and inter-eviction intervals, and a small dictionary of 1600 keywords in a constrained remote client network setup (2ms latencies, 100Mbps bandwidth). Throughput decreases with increasing number of query keywords. Note that overall throughputs are roughly 50% of the case of SSDs. Naturally, smaller databases perform better.

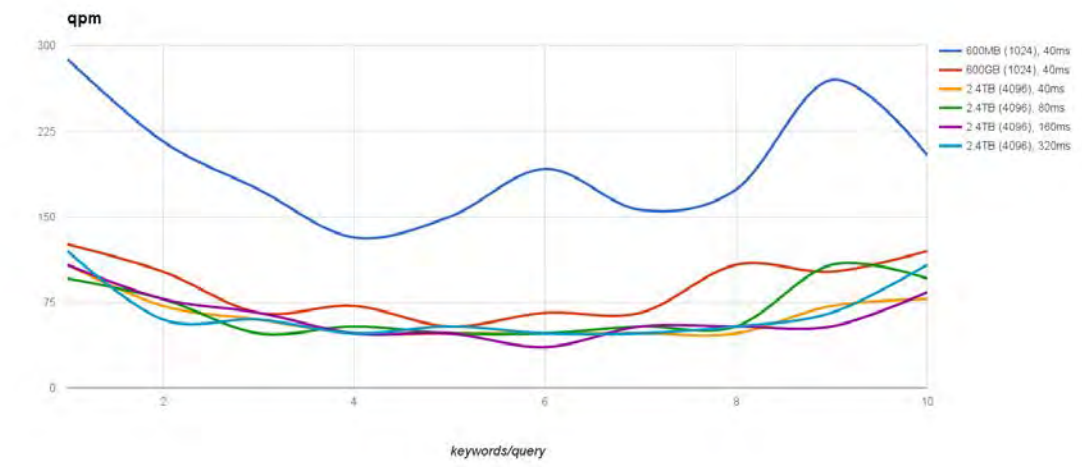

Figure 19. End-to-end performance of an aggregated, asynchronous, 3 SCPU-based Boolean keyword search query interface on databases of different sizes, block sizes, and inter-eviction intervals, and a small dictionary of 1600 keywords in a constrained remote client network setup (2ms latencies, 100Mbps bandwidth) and the data being hosted on rotational hard disks (HDDs). Throughput decreases with increasing number of query keywords. Note that overall throughputs are roughly 50% of the case of SSDs.

Similarly, also for data being hosted on rotational 7200rpm 1TB Hard Disk Drives (HDDs), Figure 20 represents the end-to-end performance of an aggregated, asynchronous, 4 SCPU-based Boolean keyword search query interface on a 3.2TB database with 4096 byte blocks, a 320ms inter-eviction interval, and a small dictionary of 1600 keywords in a constrained remote client network setup (2ms latencies, 100Mbps bandwidth. Throughput decreases with increasing number of query keywords. Note that overall throughputs are roughly 40% of the case of SSDs.

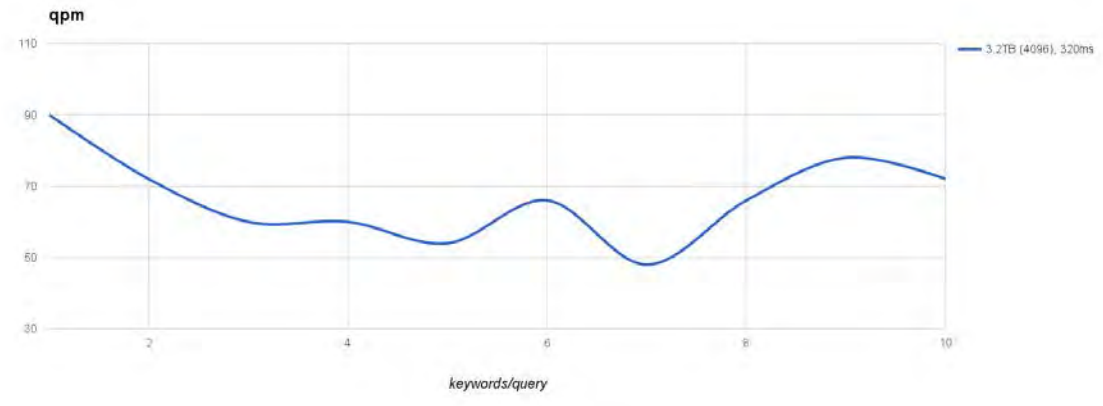

Figure 20. End-to-end performance of an aggregated, asynchronous, 4 SCPU-based Boolean keyword search query interface on a 3.2TB database with 4096 byte blocks, a 320ms inter-eviction interval, and a small dictionary of 1600 keywords in a constrained remote client network setup (2ms latencies, 100Mbps bandwidth) and the data being hosted on rotational hard disks (HDDs). Throughput decreases with increasing number of query keywords. Note that overall throughputs are roughly 40% of the case of SSDs.

4.3 Research Roadmap

A number of considerations have emerged that could shape future research.

4.3.1 SCPUs.

This seedling has conclusively shown that SCPUs can be successfully deployed as **secure and efficient anchors of trust** in systems enforcing security properties such as access privacy. We strongly believe that software-only cryptography-only approaches are bound to fail as building blocks for any practically deployable end-to-end solution until a significant crypto breakthrough delivers orders of magnitude faster new trapdoors. Until then, as also discussed repeatedly elsewhere in this project, operations inside trusted SCPUs are going to be 3-4 orders of magnitude cheaper than the equivalent trapdoor-based cryptography (whether homomorphisms, or other crypto constructs).

4.3.2 Policy Enforcement.

Finally, one of the significant advantages of using SCPUs in this context is the ability to execute arbitrary logic that constraints the data path according to e.g., given query policies that may restrict e.g., certain query patterns but not others, or based the decision on factor such as whether the submitted query keywords are relevant to a given, currently active context or not. An ORAM-based query model may not immediately show this significant advantage over any traditional software/crypto-only approaches.

4.3.3 Next Generation Trusted Hardware.

Unfortunately, existing SCPU hardware is not built for high throughput data processing. Its crypto engines feature high latencies, its I/O channels are not optimized for high speed data paths and its CPUs are significantly slower than existing untrusted x86 hardware. In the long-term it may be important to design and build a new generation of trusted hardware platforms that are cheaper, more streamlined and adaptable. In Private Machines we are working on the ENFORCER platform which provides significant compute and I/O capabilities over existing SCPUs at a fraction of the cost.

4.3.4 New Streamlined Certification Models.

It is also important to align the certification models that are used to evaluate the logical and physical security of SCPUs with the realities in the field and the actual requirements of the target deployments. The current one-size-fits-all model, while allowing for certain flexibility (e.g., variable security policy), contains a number of arcane requirements and concepts that are not applicable to the outsourced computation scenario and in fact may result in a Trusted Computing Base (TCB) increase and overall decreased security level. The Department of Defense (DoD) may choose to devise a set of new certification standards for SCPUs in the context of clouds and general outsourced deployments with multiple mistrusting clients.

4.3.5 Cost Models.

It is essential to design and deploy consistent metrics to compare both security and efficiency of different systems across multiple design and deployment dimensions. We believe a good set of building blocks are the actual $ costs associated with computing primitives: CPU cycles, network transfer of a bit, and the storage of a bit at different latency points. Such building blocks can then

be assembled into complex models evaluating the dependent security/$, security/throughput and throughput/$ quantities, among many others. Ultimately, for a given assurance dimension and metric of security it would be desirable to have the ability to meaningfully compare even radically different solutions.

4.3.6 Indexing Structures.

HS-ORAM is deploying a posting-list based approach in which the indexing ORAM **B-IDX:ORAM** stores the index which maps search keywords (or, more precisely, keyword IDs) to their corresponding posting lists, the set of block IDs that contain the corresponding keyword. This layout has significant drawbacks since it limits the throughput of inbound documents at indexing time where heaving indexing work is performed on each individual keyword/document combination added. For access privacy, this results in the firing of an ORAM transaction for each access to the corresponding posting lists which overall limits the throughput to about 2-15kbps for documents added the first time. Several solutions exist to make this more efficient. (1) Devise bulk-loading mechanisms that bypass the ORAM layer and can function effectively without privacy to bulk-load large amounts of data. (2) Optimize the posting-lists based mechanisms by e.g., allowing append operations with full privacy. (3) Design new advanced specialized oblivious indexing mechanisms that may or may not be based on ORAM as an underlying primitive.

4.3.7 Hard Disk Drive vs. Solid State Drive Media.

Storage media is essential since it originates the data and impacts individual client latencies. SSDs seem to perform 2x better than rotational 7200rpm HDDs.

5.0 CONCLUSION

This report has shown that SCPUs can be successfully deployed as anchors of trust in systems enforcing security properties such as access privacy. Results suggest significant improvements of up to 2 orders of magnitude over existing work, especially for large data sets, complex queries, and scenarios requiring the enforcement of query- and content-based query policies beyond simple access control.

6.0 REFERENCES

[1] Rakesh Agrawal, Dmitri Asonov, Murat Kantarcioglu, and Yaping Li, "Sovereign Joins", in Ling Liu, Andreas Reuter, Kyu-Young Whang, and Jianjun Zhang, editors, pp. 26-39, *International Conference on Data Engineering,* 2006.

[2] Yao Chen and Radu Sion. "On the (Im) Practicality of Securing Untrusted Computing Clouds with Cryptography", URL: http://digitalpiglet.org/research, Accessed May 21, 2015.

[3] Yao Chen and Radu Sion, "To Cloud or Not To", URL: http://digitalpiglet.org/research, Accessed May 21, 2015.

[4] Yao Chen and Radu Sion, "On securing untrusted clouds with cryptography", *9th annual ACM workshop on Privacy in the electronic society*, pp. 109–114, 2010.

[5] Rosario Gennaro, Craig Gentry, and Bryan Parno, "Non-interactive verifiable computing: Outsourcing computation to untrusted workers", *30th International Cryptology Conference*, pp. 465–482, 2010.

[6] Michael Gertz and Sushil Jajodia, Handbook of Database Security: Applications and Trends, Springer Verlag, New York, First Edition, 2008.

[7] HweeHwa Pang and Arpit Jain and Krithi Ramamritham and Kian-Lee Tan, "Verifying Completeness of Relational Query Results in Data Publishing", *24th ACM SIGMOD International Conference on Management of Data / Principles of Database Systems*, pp. 407-418, 2005.

[8] Murat Kantarcioglu and Chris Clifton, "Security issues in querying encrypted data", *Working Conference on Data and Applications Security and Privacy*, pp. 325–337, 2005.

[9] Luc Bouganim and Philippe Pucheral, "Chip-secured data access: confidential data on untrusted server", *28th international conference on Very Large Data Bases*, pp. 131–141, 2002.

[10] Maithili Narasimha and Gene Tsudik, "DSAC: integrity for outsourced databases with signature aggregation and chaining", *14th ACM international conference on Information and knowledge management*, pp. 235–236, 2005.

[11] Einar Mykletun and Gene Tsudik, "Incorporating a secure coprocessor in the database-as-a-service model", *Innovative Architecture on Future Generation High-Performance Processors and Systems*, pp. 38–44, 2005.

[12] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan, "Fully homomorphic encryption over the integers", *29th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 24 - 43, 2010.

[13] Emil Stefanov, Elaine Shi, and Dawn Xiaodong Song, "Towards practical oblivious ram", *19th Annual Network & Distributed System Security Symposium*, pp. 27-39, 2012.

[14] Emil Stefanov, Marten van Dijk, Elaine Shi, Christopher W. Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas, “Path oram: an extremely simple oblivious ram protocol”, *ACM Conference on Computer and Communications Security*, pp. 299–310, 2013.

LIST OF ACRONYMS

AES.....Advanced Encryption Standard

ATM.....Automated Teller Machine

CBC.....Cipher Block Chaining

CPU.....Central Processing Unit

CTR.....Counter Mode

DoD.....Department of Defense

DRAM.....Dynamic Random Access Memory

HDD.....Hard Disk Drive

HS-ORAM.....High Speed Oblivious Random Access Memory

HTTP.....HyperText Transfer Protocol

IARPA.....Intelligence Advanced Research Project Activity

I/O .....Input/Output

iSCSI.....Internet Small Computer System Interface

MLS .....Multi-Level Security

MPC.....Multi-Party Computation

ORAM.....Oblivious Random Access Memory

RNG.....Random Number Generator

SATA.....Serial Advanced Technology Attachment

SCPU.....Secure Processor

SSD .....Solid State Drive

TCB.....Trusted Computing Base